NUI Galway
OÉ Gaillimh

Semester 1 Examinations 2019-20

Exam Code(s)	4BCT, 4BS, 1EM, 1MDM
Exam(s)	B.Sc. (CS&IT) B.Sc. Erasmus MA (Digital Media)
Module Code(s)	CT404, CT336
Module(s)	Graphics and Image Processing
Paper No.	
Repeat Paper	
External Examiner(s)	Dr. J. Howe
Internal Examiner(s)	Prof. M. Madden * Dr. S. Redfern
Instructions:	Answer any three questions. All questions carry equal marks.
Duration	2 hours
No. of Pages	7
Discipline(s)	Computer Science
Course Co-ordinator(s)	

Requirements:

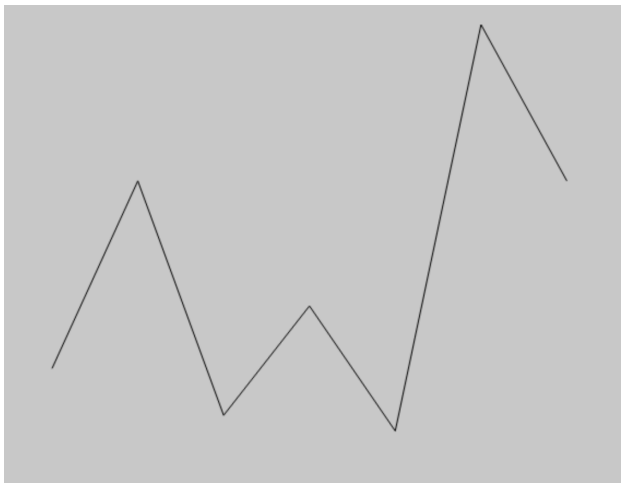
MCQ — Release to Library: Yes [X] No [ ]
Handout
Statistical/ Log Tables
Cambridge Tables
Graph Paper
Log Graph Paper
Other Materials
Graphic material in colour — Yes [x] No [ ]

PTO

Q.1. (Graphics)

(i) Provide short sections of code illustrating **translation**, **rotation**, and **scaling** in either Canvas2D or Threejs. Note that the final page of this exam paper lists some commonly used functions in Canvas2D and Threejs. [10]

(ii) Using the code below as a starting point, write Javascript/Canvas2D code for use in the **draw()** function which will draw a line graph from the numbers contained in the **data[]** array. The graph should apply appropriate scales on the *x* and *y* axes, bearing in mind that the values in **data[]** may be changed, i.e. you should not hard-code the scales. There is no requirement to label the axes. [10]

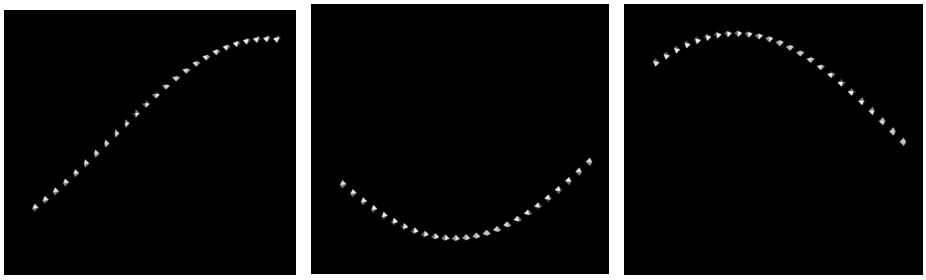


```
<html>
  <head>
    <script>
    function draw() {
        var canvas = document.getElementById("canvas");
        var ctx = canvas.getContext("2d");
        var data = [6, 18, 3, 10, 2, 28, 18];

        // to do: write code here to draw a line graph using Canvas2D

    }
    </script>
  </head>

<body onload='draw();'>
  <canvas id="canvas" width="600" height="450"></canvas>
</body>
</html>
```

Q.2. (Graphics)

(i) Using the Javascript/Threejs code provided below (and on the next page) as a starting point, add code in the **draw()** and **animate()** functions as indicated by `// TO DO`.

The program should produce an animation similar to that depicted below (3 images, each of which is a frame from the animation), whereby `Math.Sin(angle)` is used to change the *y* component of the position of each mesh, with the value of *angle* changing over time. [10]



(ii) Add code so that the camera rotates clockwise/anticlockwise around the world's *y* axis as long as the left/right arrow keys are held, while continuing to have it look at the world's origin (x=0, y=0, z=0). Hint: the keyCode values for the left/right arrow keys, as received by the onkeydown event in Javascript are 37 and 39. [10]

```
<html>
 <head>
  <script src="three.js"></script>
  <script>
    var scene, camera, renderer;
    var meshes = [];
    var angle = 0;

    function draw() {
      // create renderer attached to HTML Canvas object
      var c = document.getElementById("canvas");
      renderer = new THREE.WebGLRenderer({ canvas: c });

      // create the scenegraph
      scene = new THREE.Scene();

      // create a camera and a light
      camera = new THREE.PerspectiveCamera(75, 1, 0.1, 1000);
      camera.position.z = 100;
      var light = new THREE.PointLight(0xffffff);
      light.position.set(10, 0, 25);
      scene.add(light);

      // define a material and geometry shared by each Mesh
      var material = new THREE.MeshLambertMaterial({color: 0xf
fffff});
      var geometry = new THREE.OctahedronBufferGeometry(2);
```

[CODE LISTING CONTINUES ON NEXT PAGE]

```
      // add a set of Meshes to the scene
      for (var xpos=-60; xpos<=60; xpos+=5) {
          // TO DO (1 of 2): create a THREE.Mesh using the
geometry and material defined above
          // Position this Mesh at x=xpos, y=0, z=0
          // And also add each mesh to the meshes[] array
      }

      animate();
    }

    function animate() {
       setTimeout(animate, 1000/60);
       angle += 0.1;
       /* TO DO (2 of 2): change the y component of the
position of each object in the meshes[] array,
using Math.Sin(angle) which takes a number (angle) and
     returns a value from -1 to +1 */

         renderer.render(scene, camera);
    }
  </script>
 </head>

 <body onload="draw();">
   <canvas id="canvas" width="600" height="600"></canvas>
 </body>
</html>
```

Q.3. (Graphics)

(i) Many of the techniques used in real-time 3D graphics programming attempt to maximise the realism of the rendered scene while processing a minimal number of polygons. With specific reference to the so-called 'polygon budget', and using diagrams where appropriate, discuss each of the following five techniques: [15]

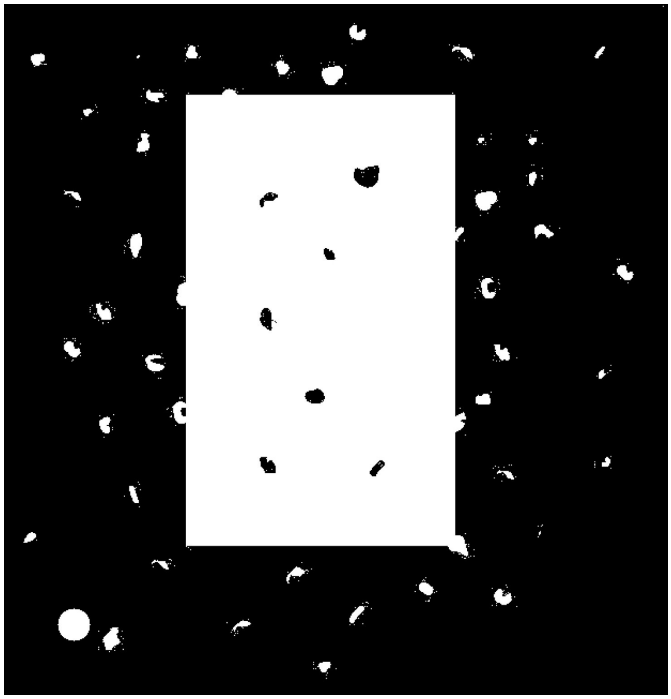
a) Frustum Culling
b) Bump Mapping
c) Back Face Culling
d) Billboards
e) Levels-of-Detail (LODs)

(ii) Discuss the Flat Shading, Gourard Shading and Phong Shading algorithms, illustrating each with a diagram. [5]

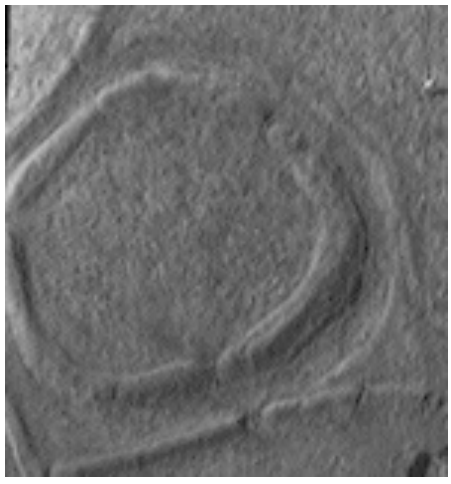
Q.4. (Image Processing)

(i) Describe the *mathematical morphology* approach to image processing. Outline some typical circumstances in which this approach is useful. [4]

(ii) The image below contains a large white rectangle and a number of patches of noise. Outline and defend a morphology-based algorithm for automatic isolation of the rectangle. Use sketches to indicate approximately how the image would appear following each step of your solution. [7]



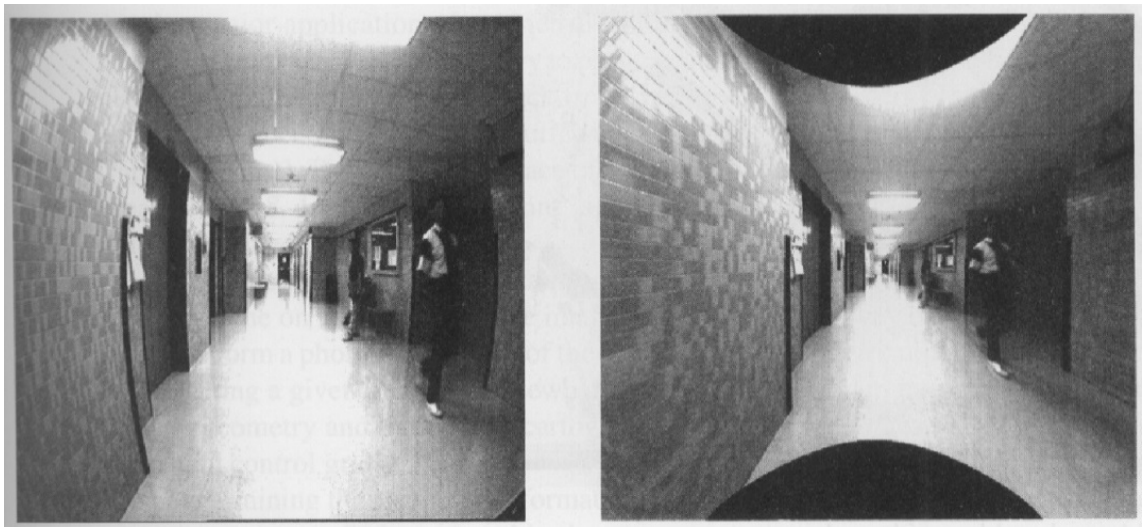
(iii) Outline, at a high level, a suitable algorithm for extracting 3D information from a stereo pair of images, such as the image pair illustrated below. [9]



Q.5. (Image Processing)

(i) Camera decalibration is a technique for geometric correction of images which is often employed when sources of geometric error are poorly understood. With regard to camera decalibration:

- Outline the use of reference images such as grids of dots to construct and apply geometric corrections to images captured with a wide-angle lens (e.g. the image below). Use the terms 'control points', 'pixel filling', and 'bilinear interpolation' in your answer. [7]
- Explain why you would expect a reference image to be constructed with black markings on a white background (or white markings on a black background) [3]



(ii) Convolution (also called 'filtering') is one of the most fundamental low-level image processing techniques.

- Describe, with use of a diagram, the convolution algorithm. [3]
- Present a convolution kernel suitable for noise reduction. Explain how this kernel produces the desired result, assuming a greyscale image with pixel values in the range 0 to 255. [3]
- Present a convolution kernel suitable for edge detection. Explain how this kernel produces the desired result, assuming a greyscale image with pixel values in the range 0 to 255. [4]

Some useful methods/properties of the Canvas 2D Context object:

Method/Property	Arguments/Values	Notes
fillRect	(Left, Top, Width, Height)	Draw a filled rectangle
beginPath	None	Start a stroked path
moveTo	(X, Y)	Move the graphics cursor
lineTo	(X, Y)	Draw a line from graphics cursor
stroke	None	End a stroked path
fillStyle	="rgb(R,G,B)"	Set fill colour
strokeStyle	="rgb(R,G,B)"	Set line colour
save	None	Save the current coordinate system
restore	None	Restore the last saved coord system
translate	(X,Y)	Translate the coordinate system
rotate	(angle)	Rotate the coordinate system clockwise, with angle in radians
scale	(X,Y)	Scale the coordinate system independently on the X and Y axes

Some useful objects/methods from the Threejs library:

Object/Method	Notes
new THREE.WebGLRenderer({canvas:c})	Constructs a renderer, attached to the Canvas object c
new THREE.PerspectiveCamera(fov,aspect,near,far)	Constructs a camera, with the specified field-of-view, aspect ratio, near clipping distance, far clipping distance
new THREE.Scene();	Constructs a scene
new THREE.PointLight(0xffffff);	Constructs a white point light
object.position.set(x,y,z)	Sets an object's x,y,z position relative to its parent
object.rotation.set(x,y,z)	Sets an object's x,y,z rotation (using Euler angles) relative to its parent
object.rotateOnAxis(new THREE.Vector3(0,1,0), 0.1)	Rotates object by 0.1 radians on the y axis
object1.add(object2)	Sets object2 as a child of object1
object.parent	Obtains a reference to the parent of object
camera.lookAt(new THREE.Vector3(0,0,0));	Turns a camera object to face the world coordinate 0,0,0
new THREE.BoxGeometry(20, 20, 20)	Constructs Box geometry, with specified width, height, depth
new THREE.MeshLambertMaterial({color: 0xfd59d7})	Constructs a Lambert (Phong) material of the specified colour
new THREE.Mesh(geometry, material)	Constructs a mesh using the specified geometry and material
renderer.render(scene, camera)	Uses a renderer to draw a scene as seen by a camera, onto the renderer's Canvas